DESIGN AND IMPLEMENTATION OF A MULTI-AGENT SYSTEMS LABORATORY

A Thesis
Presented to
The Academic Faculty

by

Malachi G. Jones

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
May 2009

DESIGN AND IMPLEMENTATION OF A MULTI-AGENT SYSTEMS LABORATORY

Approved by:

Professor Jeff Shamma, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Magnus Egerstedt
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Eric Feron
School of Aerospace Engineering
Georgia Institute of Technology

Date Approved: May 12 2009

To my mom, brother, grandma, and extended family for all their support and encouragement.

TABLE OF CONTENTS

DEDICATION iii

LIST OF FIGURES vii

SUMMARY ix

I INTRODUCTION 1

1.1 Motivation 1

1.2 Multi-Agent Systems Laboratory Components 2

1.2.1 NCS 2

1.2.2 Communication 2

1.2.3 Localization 3

1.2.4 Navigation 3

1.3 Overview of Thesis 4

II LABORATORY HARDWARE INVESTIGATION 5

2.1 NCS hardware 5

2.1.1 Khepera features 6

2.1.2 Khepera limitations 7

2.2 Localization system hardware 8

2.2.1 Overview of the localization systems 8

2.2.2 Analysis of localization systems 11

2.2.3 Vicon motion capture system details 12

III DESIGN 14

3.1 Communication 14

3.1.1 Communications protocol 14

3.1.2 Network Architecture 16

3.2 Navigation 19

3.2.1 Path Planning 19

3.2.2 Motion Control 20

3.3 Localization 23
3.3.1 Motion Capture Configuration 24
3.3.2 Motion Capture Interface 25
3.3.3 Agent Interface 26
3.4 Summary of design 28
IV TESTING 29
4.1 Individual Component Testing 29
4.1.1 Communication 29
4.1.2 Navigation 30
4.1.3 Localization 33
4.2 Integrated Component Testing 34
4.2.1 Gossip algorithm 34
4.2.2 How the algorithm tests the infrastructure 34
4.2.3 Test Procedures 34
4.2.4 Results and analysis 35
V LABORATORY USERS MANUAL 36
5.1 Equipment Setup and Configuration 36
5.2 Vicon Motion Capture System 37
5.2.1 Initializing the System 37
5.2.2 Calibration 38
5.2.3 Creating agent tracking objects 41
5.3 Server 43
5.4 Khepera 44
5.4.1 Developing Software 45
5.4.2 Transferring and Executing script 46
VI CONCLUSION 47
6.1 Summary 47
6.2 Future Recommendations 47

APPENDIX A SOURCE CODE LIBRARY 51

LIST OF FIGURES

1 Khepera mobile embedded device 6
2 Cricket unit 8
3 CMUcam vision sensor hardware 9
4 Motion capture studio 10
5 Vicon system components 12
6 Vicon motion capture system architecture 13
7 Data format 15
8 Data Packet example 16
9 Wireless network types 17
10 Agent rotation 21
11 Localization flow chart 23
12 symmetric versus asymmetric patterns 24
13 Triangular marker pattern on a khepera 25
14 Vicon and GPS servers 25
15 Agent interface 27
16 GPS format 28
17 Communications test between two agents 30
18 Motion capture redesign 32
19 Vicon iQ software interface 37
20 Camera output screen 38
21 Calibration screen 39
22 Calibration screen 40
23 Setting the global reference point 40
24 Setting the global reference point 41
25 Select agent object markers 42
26 Agent object created 42
27 Khepera_GPS solution in Visual Studio 2008 43

28 Localization data output 44
29 Code Blocks (Khepera IDE) 45
30 Compiler option display 45
31 Secure shell host client 46

SUMMARY

This thesis presents the design, development, and testing of a multi-agent systems laboratory that will enable the experimental investigation of Networked Control Systems. Networked Control Systems (NCS) are integrations of computation, networking, and physical dynamics, in which embedded devices are networked to sense, monitor, execute collaborative tasks, and interact with the physical world. As the potential for applications of NCS has increased, so has the research interest in this area. Possible applications include search and rescue, scientific data collection, and health care monitoring systems. One of the primary challenges in applying NCS is designing distributed algorithms that will enable the networked devices to achieve global objectives. Another challenge is in ensuring that distributed algorithms have the necessary robustness to achieve those global objectives in dynamic and unpredictable environments. A multi-agent systems laboratory provides the researcher with a means to observe the behavior and performance of distributed algorithms as they are executed on a set of networked devices. Through this observation, the researcher may discover robustness issues that were not present in computer simulation. The objective of this research is to design and implement the infrastructure for a multi-agent systems laboratory to observe distributed algorithms implemented on networked devices.

CHAPTER I

INTRODUCTION

1.1 Motivation

A multi-agent system is comprised of a set of agents that are assumed to be rational, intelligent, and autonomous [1]. Agents can communicate and interact with each other to achieve a global goal. Since the agents are autonomous, it may be possible to achieve this goal if a subset of the agents fail or malfunction. Multi-agent systems are well suited for problems that are too complex or impossible to solve using a monolithic system, e.g. search and rescue [2] and scientific data collection [3]. Distributed algorithms can be developed to deploy onto multi-agent systems to solve these types of problems.

Distributed algorithms may be executed in environments that are highly volatile, dynamic, and unpredictable. This motivates a desire for the algorithms to be very robust. Computer simulation can be a useful tool for analyzing this robustness, but there may be test cases not included in the simulation that could help improve the analysis. A multi-agent systems laboratory provides a means for observing the behavior of these distributed algorithms in a physical environment, which may allow for the discovery of robustness issues not present in the simulation. One can then use this discovery to further improve the accuracy of a simulation. In some instances, however, it may be very difficult or impossible to simulate these robustness tests. Instead, the multi-agent systems laboratory can be used to observe and analyze these test cases.

1.2 Multi-Agent Systems Laboratory Components

The four fundamental components of a multi-agent systems laboratory are : i) NCS, ii) communication, iii) localization, and iv) navigation. NCS is a physical manifestation of the multi-agent system and is composed of a networked set of mobile embedded devices. One of the primary goals of the laboratory is for the embedded devices to perform collaborative tasks, which require that each device has a mechanism to communicate with neighboring devices. The collaborative tasks of interest, which include area coverage, also require localization and navigation capabilities in addition to communication.

1.2.1 NCS

A Networked Control System (NCS) is defined as an integration of computation, networking, and physical dynamics, in which embedded devices are networked to sense, monitor, execute collaborative tasks, and interact with the physical world. The khepera [4] is the embedded device that will be used to implement NCS in the laboratory. It satisfies the hardware, communication, and processing requirements necessary for implementation in NCS. In addition, the khepera also features an operating system that makes it simpler to program the embedded device to execute collaborative tasks.

1.2.2 Communication

There are several network types that can be implemented in the laboratory to facilitate the exchange of information between neighboring agents. In a centralized network, the nodes do not directly communicate with each other. Each node communicates with an intermediate device, such as an access point, and that device is responsible for sending the information to the appropriate destination node. The key advantage of this architecture is that if every node can communicate with the intermediate device, then all the nodes can communicate with each other. In contrast, a decentralized network, such as ad-hoc, requires that nodes who wish to communicate

with each other be in listening range of each other. The dynamic network topology of a multi-agent systems laboratory is one of the many reasons that an ad-hoc network is a more desirable choice.

1.2.3 Localization

A fundamental problem in this laboratory and a recurring issue in mobile devices is a need to develop a mechanism that will allow the mobile device to acquire orientation and position information with respect to some global reference point. Acquiring localization information can be more challenging in an in-door environment, such as this laboratory, because gps is unavailable. Research has been undertaken to develop hardware that can be attached to mobile devices that will provide localization information. The developed hardware includes the Cricket [5], which was researched at MIT and CMUcam vision sensor [6] that was researched at Carnegie Mellon. An alternative approach to attaching the localization hardware to the device is utilizing a stand alone system, e.g. motion tracking system, that sends localization information to the device over a network. This approach will be used for the laboratory.

1.2.4 Navigation

The process of enabling a mobile device to travel from an arbitrary location to a specified destination can be separated into two phases. Phase I consists of finding a path that allows the device to reach the destination. Complications such as static obstacles, nonholonomic constraints, and optimal path requirements can make finding this path difficult. In phase II, the goal is to guide the mobile device along the generated path by controlling the motors. Problems such as will slippage and imperfect hardware make guiding the device increasingly challenging.

1.3 Overview of Thesis

The remainder of the thesis is organized as follows. Chapter II discusses the investigation of the embedded systems and localization hardware. Requirements for each of the components of the laboratory infrastructure are presented along with the corresponding design in Chapter III. Chapter IV describes how the designs were tested and also details the results of the tests. Chapter V presents a users manual for the laboratory that covers initializing the equipment and programming the embedded device.

CHAPTER II

LABORATORY HARDWARE INVESTIGATION

Hardware for NCS and the localization system will be investigated in the forthcoming sections. Careful selection of this hardware is critical in achieving the overall objective of developing a multi-agent systems laboratory. In order to help ensure that prudent hardware selections are made, each hardware device under consideration will be evaluated using the following criteria: i) To what extent does the hardware solve the problem? ii) How difficult would it be to modify the hardware to solve the problem if the problem cannot be completely solved using the original configuration of the hardware? iii) What are the hardware limitations, and how can those limitations affect the performance of the laboratory?

2.1 NCS hardware

A set of mobile embedded devices with wireless networking capabilities are necessary to implement NCS. The small market for these mobile embedded devices severely limits the number of options that are available. Since laboratory space is limited and a large number of embedded devices are desired, available options are further reduced because the devices also need to have sufficiently small dimensions. The khepera was the only device to meet the aforementioned minimum requirements that are mobility, wireless network capabilities, and size. Features such as a powerful embedded systems processor and an on-board operating system make the khepera (see Figure 1) a solid platform to build NCS. However, there are some limitations that could affect laboratory performance.

Figure 1: Khepera mobile embedded device

2.1.1 Khepera features

The most significant and impressive features of the khepera [4] are the: i) microprocessor architecture which includes memory, ii) mobility, iii) wireless networking capabilities, and iv) operating system. These features are significant and impressive because they have the potential to dramatically enhance the performance and efficacy of NCS in the laboratory. A 400 MHz ARM microprocessor that includes 64 MB of ram provides the khepera with the processing horsepower to run most applications that are found on the standard desktop computer. In terms of the laboratory, the processing power will allow the khepera to execute complex distributed algorithms very fast. It also allows for helper applications that support the distributed algorithm to run simultaneously without dramatically diminishing performance.

Two independent servo motors allow the khepera to be highly mobile. One major benefit of this mobility is that the khepera can change its orientation without changing its position. This feature reduces the complexity of the motion controller and the path planner that will be implemented. Internal controllers within the khepera handle the low-level motor control details that include supplying the correct amount of current to achieve and maintain a set speed . These internal controllers can allow the khepera to accelerate and decelerate at a user specified rate.

An 802.11b compliant wireless network card that is on-board the khepera enables it to communicate with peers indirectly through an intermediate node or directly by modifying the network mode from infrastructure *(indirect)* to ad-hoc *(direct)*.

Although 802.11g wireless cards allow for data transmission at rates of up to 54 Mbps, the 802.11b provides a significant power savings that is very important because the kheperas are battery powered. Furthermore the rate of transmission of the 802.11b card, which is 11 Mbps, is sufficient for the communication requirements of this lab.

The on-board Linux operating system provides the software developer with powerful mechanisms that make programming the khepera simpler and more efficient. Hardware drivers are mechanisms that handle the low-level hardware interactions that include reading and writing to registers on the hardware. This allows the developer to focus on developing software for the high-level interactions that are required for the laboratory. An application programming interface (API) is a mechanism that provides the programmer with a set of routines, data structures, and object classes that facilitate the development of distributed algorithms and applications that will be executed on the khepera. Furthermore, the operating system handles memory management and application multitasking that enable the distributed algorithms to run efficiently.

2.1.2 Khepera limitations

Developmental embedded device platforms, such as the khepera, have several unpublished hardware issues, limited technical support, and documentation that is inadequate if it exists. This can severely slow down and in some cases halt the software development process needed to program these embedded devices. Unpublished hardware issues often present serious problems for the software developer as these issues can affect how the developed software is executed, thereby adversely impacting the expected performance and behavior of the device. One hardware issue was the non-functional sonar sensors. These sensors are intended to provide the khepera with long range sensing capabilities. Another hardware issue was the wireless card. On some kheperas, the on-board wireless card would periodically malfunction and prevent any

communication with other kheperas. In the event that the developer discovers a hardware issue, limited technical support makes it very difficult to resolve the issue. Since good documentation of the embedded device helps the programmer understand the underlying hardware architecture and software libraries, inadequate documentation dramatically slows down the development process as more time is spent on figuring out how the device works and is structured.

2.2 Localization system hardware

Two fundamental approaches for solving the indoor localization problem are: i) attaching localization hardware to the device and ii) transmitting localization information from a stand alone localization system to the device. Localization hardware under investigation are the Cricket [5], CMUcam vision sensor [6], and motion capture system, where the last system is stand alone and the other two systems can be attached to the device. A brief overview of each of the localization systems will be presented in section 2.2.1. The systems will then be analyzed and evaluated in section 2.2.2. In concluding the investigation, the chosen localization system will be presented and described in more detail in sections 2.2.3 .

2.2.1 Overview of the localization systems

2.2.1.1 Cricket

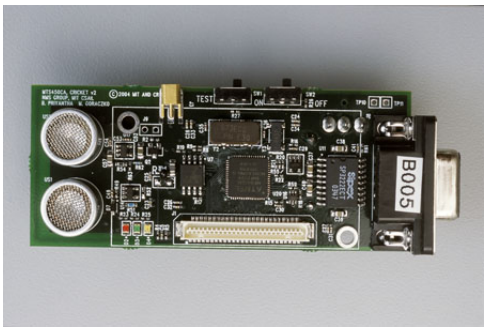
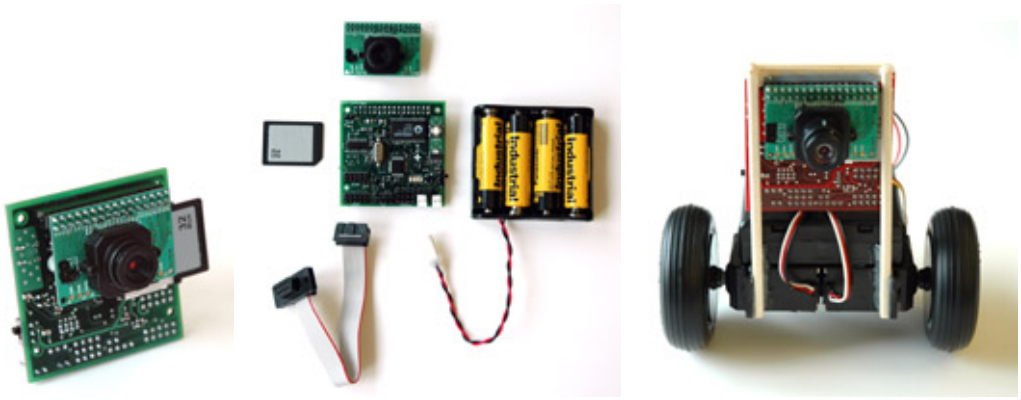


Figure 2: The cricket unit can function as either a beacon or listener.

Cricket [5] is an indoor localization system developed at MIT that provides a

host device with position and orientation information by exploiting the difference in propagation speeds between RF (speed of light) and ultrasound (speed of sound). Wall and ceiling mounted beacons as in Figure 2 broadcast information on an RF channel, while simultaneously transmitting an ultrasonic pulse. A Cricket listener (see Figure 2), attached to a host devise either through a RS232 serial connection or a compact flash interface, listens for the RF signal and the corresponding ultrasonic pulse. The listener then runs algorithms to correlate RF and ultrasound to determine the distance and orientation with respect to the beacons.

2.2.1.2 CMUcam



(a) camera **(b)** vision sensor package **(c)** sensor attached to device

Figure 3: CMUcam vision sensor hardware

The CMUcam vision system (see Figure 3) [6], developed at Carnegie Mellon, provides small embedded systems with an intelligent sensor that can be used to identify fixed global landmarks. Once global landmarks are identified, the location of the embedded system with respect to those landmarks can be determined. Landmarks should be objects that have highly contrasting and intense colors if one wishes to obtain the best performance from the sensor. Custom algorithms can be written to calculate the distance the embedded system is from the landmarks. With these distances, the global position of the device can be determined.

2.2.1.3 Motion Capture system

Figure 4: Motion of a human is studied in this motion capture studio.

A motion capture system is a tool that allows for the motion of an object or group of objects to be recorded, studied, and analyzed. The system is equipped with specialized optical cameras that can detect infrared lights that are reflected off highly reflective markers. These reflective markers are attached to the objects whose motions are being studied. By arranging the markers in unique geometric patterns, it is possible to identify the components of the object. A typical application is motion capture of the human body, including the hands, torso, and feet (see Figure 4). The features of the motion capture can be exploited to track the movement of mobile agents.

2.2.2 Analysis of localization systems

2.2.2.1 Cricket

Although Cricket possesses features that are required for the lab, e.g. scalability and precision, there is a severe hardware compatibility issue that would prohibit implementing this system in the laboratory. Because Cricket uses active beacons and passive listeners, where the embedded devices are the listeners, it is capable of operating with a large number of devices. Cricket is also capable of achieving a precision in localization information of 1 to 3 cm [5]. A significant drawback of implementing Cricket is that power is obtained from the host device to which it is attached. The embedded device that will be used for this lab, the khepera, does not offer a means for Cricket to tap into its power supply. Assuming that the khepera could be altered to enable tapping into the power supply, Cricket would drain the khepera's battery very rapidly due to its weight and power consumption. Since the khepera is a small device, it is not possible to attach a more powerful battery.

2.2.2.2 CMUcam

Unlike the Cricket, CMUcam is lightweight, has a battery supply, and is suitable for mounting unto small devices such as the khepera. However, there are two fundamental problems with implementing this system. The first problem is scalability. CMUcam relies on having a direct line of sight with fixed landmarks to determine localization information. Increasing the number of devices in the laboratory would also increase the probability that one or more landmarks would be occluded by other devices. Occlusion would adversely affect the accuracy of the localization data. The second problem is that CMUcam would require customized image processing and computer vision algorithms to be written to generate the localization information. This is outside the scope of the immediate laboratory objective.

2.2.2.3 Motion capture system

In contrast to the aforementioned systems, a motion capture systems exceeds all of the localization requirements for the lab and can be readily purchased (from several companies), setup, and configured for the laboratory. Although there are more costs associated with purchasing this type of system, the technical support, stability, and accuracy of the system more than compensates those costs. Another benefit of this hardware system is that it has the ability to track a very large number of objects, which is very important for the laboratory. After selecting the motion capture system as the localization system to use in the laboratory, a low-end system was purchased. Fortunately a few months later, a high-end motion capture system produced by Vicon [7] was donated to the laboratory.

2.2.3 Vicon motion capture system details

(a) Optical camera **(b)** Data station **(c)** calibration kit

Figure 5: Vicon system components

Vicon motion capture system [7] consists of the following hardware : i) 8 optical cameras, ii) data station, and iii) calibration kit. Each optical camera has attached hardware that emits infrared light (see Figure 5a). The cameras are able to capture the light as it is reflected off reflective surfaces,such as a marker, and convert it into a digital format. This digital information is then sent to the data station connected to the cameras. Raw digital data is requested and retrieved from the data station by a

high-end computer running the Vicon software and server. Vicon software running on the computer processes the data retrieved by the data station and is able to capture the motion of objects once a calibration procedure is completed using the software and the calibration kit (see Figure 5c). Figure 6 provides an illustration of the architecture of the system and a visual of the flow of information.

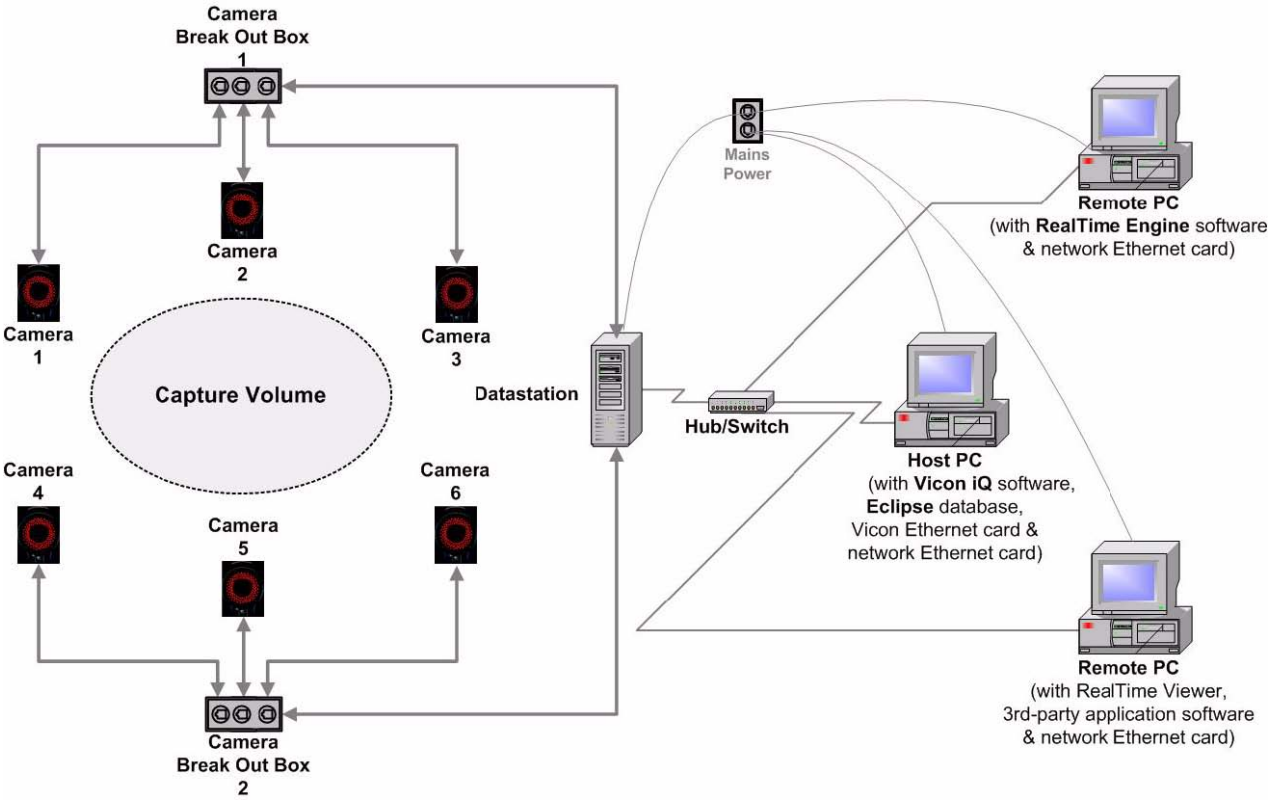


Figure 6: Vicon motion capture system architecture

CHAPTER III

DESIGN

The infrastructure for the multi-agent systems laboratory consists of three components: i) communication, ii) navigation, and iii) localization. A design was developed for the respective components. The forthcoming sections will discuss each component's design and the rationale behind why that design was chosen. A summary of the component designs will conclude the chapter.

3.1 Communication

Before any communication between agents can occur, a communications protocol and network architecture must be defined. A communications protocol is a set of agreed upon rules that describe how the agent should format information into a data packet prior to sending the information across the network. Recipients of the data can then use those rules to unpack and parse the packet to extract pertinent information. The network architecture defines the physical configuration, functional organization, and operational procedures of the network. Awareness of the network architecture provides the agent with knowledge of the appropriate procedures to follow so that information can be delivered to the desired recipients.

3.1.1 Communications protocol

Each agent wishing to send information to other agents will utilize a communications protocol with the following structure (illustrated in Figure 7). Agent identification information will be stored in the first field. The second field will store a comma delimited list of agents for which the data packet is destined. Information that an agent desires to send to respective destination agents is stored in the payload field.

More details about each field will be provided in the ensuing subsections.

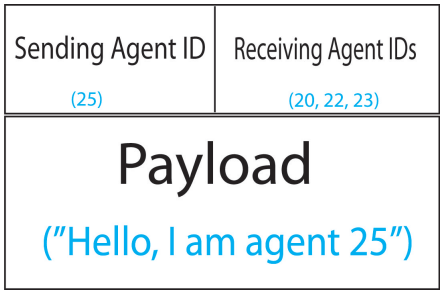


Figure 7: Data format

3.1.1.1 Sender identification number

Including a sender identification field allows the agents to avoid ambiguities that can arise when multiple agents are sending across a shared medium. Consider the following scenario. Suppose agent (R)eceiver is waiting to receive information that it has requested from agent (S)ender. Another agent, agent (A)mbiguity, has decided to request information form agent R at the same time agent S is sending a response to R also. If sender information is not included in the packets sent by S and A, agent R will not be able to decipher which data packet contains the response to its initial query to S.

3.1.1.2 Receiving agent identification numbers

There are two approaches that can be used to accomplish the task of sending information to a select group of agents. The first approach (see Figure 7), which is used in the communication protocol, is to use a delimited list in the receiving agent ID field. The sending agent broadcasts information to all the agents. Each agent checks to see whether its identification number is on the list. If an agent's identification number is not on the list, the agent ignores the message.

An alternative approach is to send a copy of that information specifically targeted to each of the agents. Figure 8 provides an illustration of this method. Observe

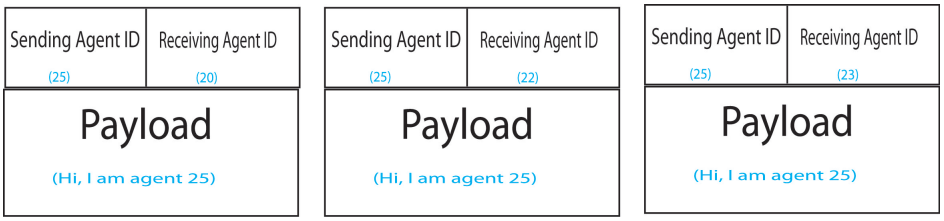


Figure 8: Illustration of method 2, which requires copies of the data packet to be individually addressed to agents 20, agent 22, and agent 23.

that this strategy requires n copies of the message to be distributed to n destination agents. The first method, in contrast, only requires one copy of the information to be sent to n agents targeted to receive the message. It is evident that the method used for the lab, method 1, utilizes the network bandwidth more efficiently.

3.1.1.3 Payload

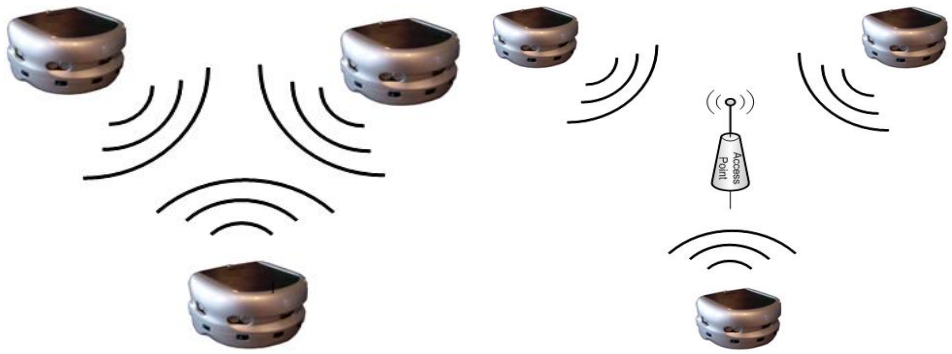
Messages that will be placed in the payload can have a maximum size of at most 500 bytes because the underlying network protocol will not transmit any larger size without fragmenting the message. Fragmentation consists of breaking the message into separate parts and then sending each part individually to the targeted destination agent. The re-assembly process required by the destination agent to assemble the separated parts into one message is undesirable in the network setup for the laboratory due to the complex and error prone nature of this process.

3.1.2 Network Architecture

Design decisions will be discussed for the following elements of the network architecture in the proceeding sections: i) network type and ii) transmission control protocol. The fundamental wireless network types are ad-hoc (*decentralized*) and infrastructure (*centralized*). In an infrastructure network, an intermediate device manages and updates a routing table and is responsible for relaying information between agents. An adhoc network, in contrast, requires each agent to create and manage its own routing table. Agents are also required to use the routing table to send and forward packets

in a manner that allow the packets to reach a destination node . Transmission control protocols outline how nodes transmit information to each other.

3.1.2.1 Network type



(a) Ad-hoc network **(b)** Infrastructure network

Figure 9: Wireless network types

An ad-hoc network allows devices to communicate directly with one another (see Figure 9a). One drawback of this configuration is that agents who wish to communicate with each must also be in listening range of each other. This can sometimes be a problem because the agents limit their transmission range to conserve power, which can prohibit agents from communicating with each other. Because of several significant drawbacks of an infrastructure configuration, an ad-hoc configuration is the more attractive choice for the multi-agent system network.

There are several drawbacks of implementing an infrastructure configuration (see Figure 9b) that prohibit this configuration from being implemented in the laboratory. In this configuration, agents can only communicate with other agents by sending the desired message to an intermediate device, e.g. access point. The access point forwards information from a sending agent to the target destination agent. The first drawback of this approach is that it introduces a single point of failure because a failure of the intermediate device will halt all communication. Second, the intermediate device decreases the network bandwidth by half. This is because a sending agent consumes bandwidth when it transmits information to the intermediate device. The

intermediate device then consumes additional bandwidth as it forwards that information to the receiving agent. Bandwidth is crucial in the laboratory as the embedded devices only support a maximum transmission rate of 11 Mbps. The third drawback, scalability, is related to the second drawback. Increasing the number of nodes in the network will dramatically amplify the loss of bandwidth due to using an intermediate device to forward the information to the receiving nodes.

3.1.2.2 Transmission control protocol

A brief overview of the two transmission control protocols will be provided. This will be followed by a discussion that will present which protocol was chosen for the lab and include an analysis of why the protocol was chosen. The first protocol to be presented is the user datagram protocol (UDP). UDP is a connectionless protocol. A connectionless protocol allows agents to transmit information to each other without first making an explicit arrangement. It is analogous to beginning a conversation with a friend before establishing eye contact and greeting that person with a hello and a handshake. UDP is a simple and light weight protocol that does not provide services such as error correction and reliable delivery of data.

TCP/IP (Transmission control protocol over Internet protocol) also referred to as simply TCP, is the compliment to UDP in that it is a connection-oriented protocol. Before messages are sent between two nodes, an elaborate three way handshake process is done by the nodes to explicitly announce that the two nodes will begin communicating with each other. One of the important services that TCP offers is the guaranteed delivery of data. This feature works by requiring the node receiving data to send an acknowledgment packet to the sender confirming that the message was received. Other services of TCP include congestion control and ordered delivery of data packets.

It may seem that TCP should be the clear choice as the transmission protocol for

the network in the lab because of the services it provides. However, closer analysis will reveal that UDP is the better choice. Guarantee delivery of data offered by TCP may not be a useful service because agents send time sensitive and/or real time data across the network. If time sensitive data is not received by the targeted nodes after the first time it is sent, the TCP protocol requires that the data is resent after a timeout interval has expired. The resent time sensitive data may no longer be of any value to the agent. Furthermore, guaranteed delivery consumes a significant amount of bandwidth because agents are required to send an acknowledgment packet upon successfully receipt of information.

3.2 Navigation

Agents may be required to travel to a desired destination from their current position. This can be achieved by first utilizing a path planner to generate a feasible path that will allow an agent to reach a destination. In order to reduce the complexity of the path planner, the presence of static and/or mobile obstacles will not be considered. A motion controller will then steer the agent along the path and adjust the agent's velocity as necessary.

3.2.1 Path Planning

Since obstacles will not be considered, a simple path planner will be sufficient for the purposes of the laboratory. The path planner will calculate the orientation angle needed to intersect the target destination while moving in a straight line. It will also continuously calculate the Euclidean distance between the current location of the agent and the target destination. An interface was developed to allow the path planner to send the calculated orientation angle and Euclidean distance to the motion controller.

3.2.2 Motion Control

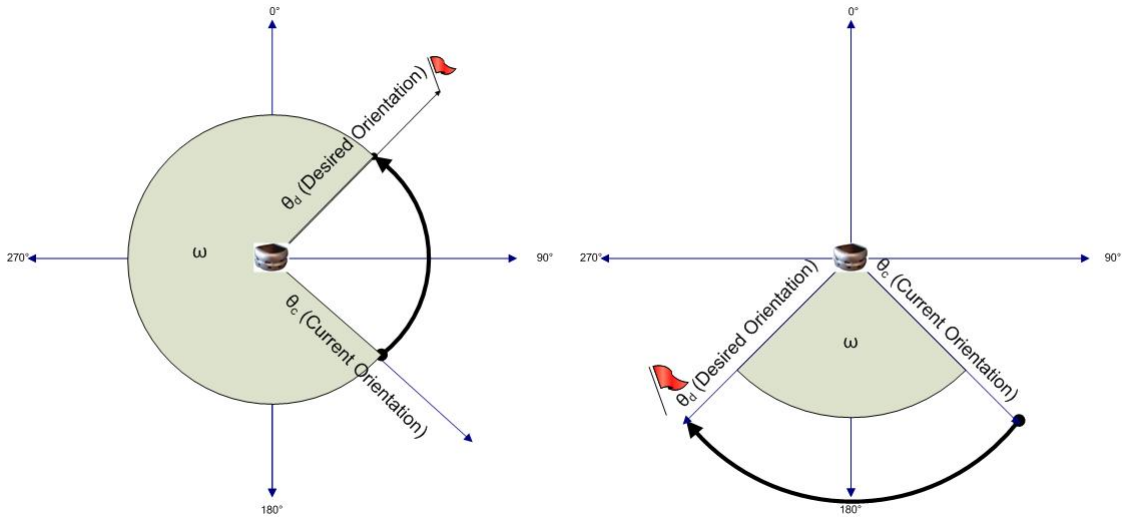
It is assumed that the motion controller will be continuously supplied with the orientation and position information of the agent. The controller is unaware of the details of how this information is being obtained. It will use the information to steer the agent and also rotate the agent to a desired orientation. Since the agent has two independently controlled motors, steering and rotation can be achieved by choosing appropriate velocities for each of the motors. As an example, rotating the agent to the left could be achieved by setting the velocity v_r of the right motor to a positive value and the value of the left motor to $-v_r$ ($v_l = -v_r$).

Low level motion control details will not be considered in the design. These details include achieving and maintaining a set velocity and controlling the agents acceleration. The agents possess internal controllers that handle this low level functionality. However, an application programming interface (API) is provided that includes mechanisms that allow for high-level motion control. The primary high-level mechanism used in the motion controller is an API function that allows for setting the khepera's velocity.

Because a simple path planner will be used, only motions that are necessary for traversing paths generated by this planner are considered. These motions are rotating the agent to a specified orientation and steering an agent in a straight line. As the agent proceeds toward the destination, it will be necessary to adjust the velocity throughout the traversal of the path. This will ensure that the agent arrives at the destination in a reasonable time period while not overshooting the target.

A simple design one can consider that will allow the agent to reach a desired orientation would be to rotate the agent in a default direction by setting the motors to constant but opposite velocities ($v_l = -v_r$). As the agent is rotating, the controller compares the difference between the current and desired orientation. Once the difference is within some threshold range, a stop command is issued to the two motors.

Now, the agent is within some acceptable error of the desired orientation. A few modifications can be made to this design to improve it dramatically.



(a) $\omega > 180\,\text{deg}$ **(b)** $\omega < 180\,\text{deg}$

Figure 10: Faster for agent to rotate counterclockwise in Figure 10a and clockwise in Figure 10b

The time it takes for an agent to reach a desired orientation can be improved by choosing an appropriate direction for the agent to rotate, instead of defaulting to an arbitrary direction. This direction is determined by considering the angle ω formed by the agents current orientation and the target orientation in a clockwise manner (see Figure 10). If ω is less than 180 degrees, the agent can reach the target orientation faster by rotating clockwise as demonstrated in Figure 10b. Conversely, if ω is greater than 180, it is better for the agent to rotate in a counter clockwise manner (see Figure 10a) to arrive at the destination orientation .

Another modification that can be made to the design is to choose a velocity that will allow the agent to reach the target destination in a minimum amount of time. An approach, based on an optimum strategy of maximum acceleration and maximum deceleration [8], would be to accelerate the agent at a maximum rate until it achieves its maximum velocity. Once the agent is within sufficient distance of the target, the agent would decelerate at a maximum rate and stop when target destination is

reached. Due to the difficulty in developing a dynamic model for the khepera, this approach is infeasible. Instead, a suboptimal strategy in determining the velocity can be considered. The velocity will be proportional to the current distance needed to reach the destination. The speed will decrease linearly as the Euclidean distance between the agent's current location and the target destination decreases.

3.3 Localization

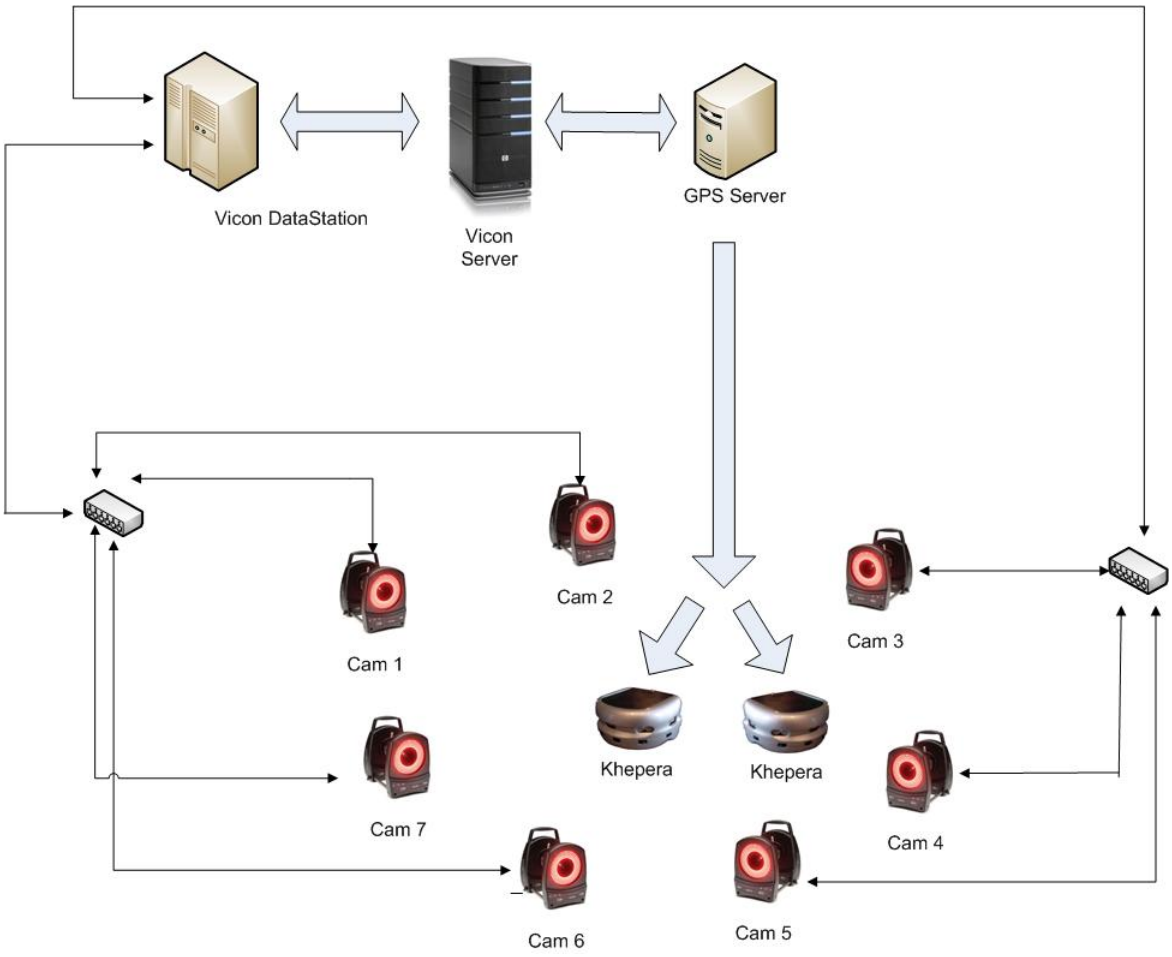
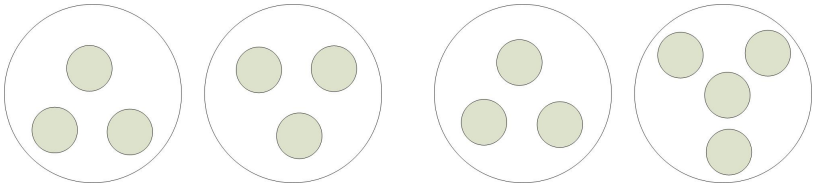


Figure 11: Localization flow chart

The localization component of the laboratory infrastructure involves the integration of three different hardware components as shown in Figure 11: i) Vicon DataStation, ii) Vicon server, iii) GPS server, and iv) khepera robots. Since the motion capture system is primarily designed to capture the motion of a small group of objects, a design configuration will be developed to allow the system to track a large number of agents. After the system has been configured to track agents, a motion capture interface will be designed to allow the GPS server to access and retrieve positioning data from the Vicon Server. Once the positioning data is processed, an agent interface will allow the GPS server to send this information to each of the agents.

3.3.1 Motion Capture Configuration



(a) Symmetric patterns **(b)** Asymmetric patterns

Figure 12: Patterns in Figure 12a are symmetric because if one of the patterns is rotated by 90 degrees, both patterns will be identical.

Although the default motion capture configuration allows for tracking of a small group of objects, it will be necessary to develop a design configuration that will allow the system to track 30 agents. When using the system to capture the motion of humans, each body part, e.g. torso, hands, and feet, can be distinguished by a unique geometric arrangement of the reflective markers. In contrast to humans, agents have a significantly smaller surface area that reduces the number of unique geometric patterns that can be placed on them. Another complication to the above constraint is that it is necessary for the geometric patterns to be asymmetric (see Figure 12b). If symmetric patterns (see Figure 12a) were allowed, the system would be unable to uniquely distinguish them.

In an ideal scenario, each agent would have a unique asymmetric geometric marker pattern. This would allow the software to always uniquely distinguish the agent and thus provide agents with the correct localization information regardless of the initial configuration of the agents upon startup of the system. Because of the limited surface area available to place the reflective makers on the agents, another strategy must be considered. Upon experimental investigation, it has been established that the Vicon software is capable of tracking multiple objects with identical geometric marker patterns. This feature will be exploited in the design.

Three reflective markers forming a triangle will be placed on the top surface of each of the agent in a manner shown in Figure 13. Since the system cannot uniquely

Figure 13: Triangular marker pattern on a khepera

identify the agents, software has been developed to allow the researcher to manually create a mapping that relates the objects that are detected by the system to the agents in the lab. This mapping will take place upon startup of the system. Before startup, each of the agents will be set to a home orientation because the system designates the agents orientation at startup as the 0 degree orientation.

3.3.2 Motion Capture Interface

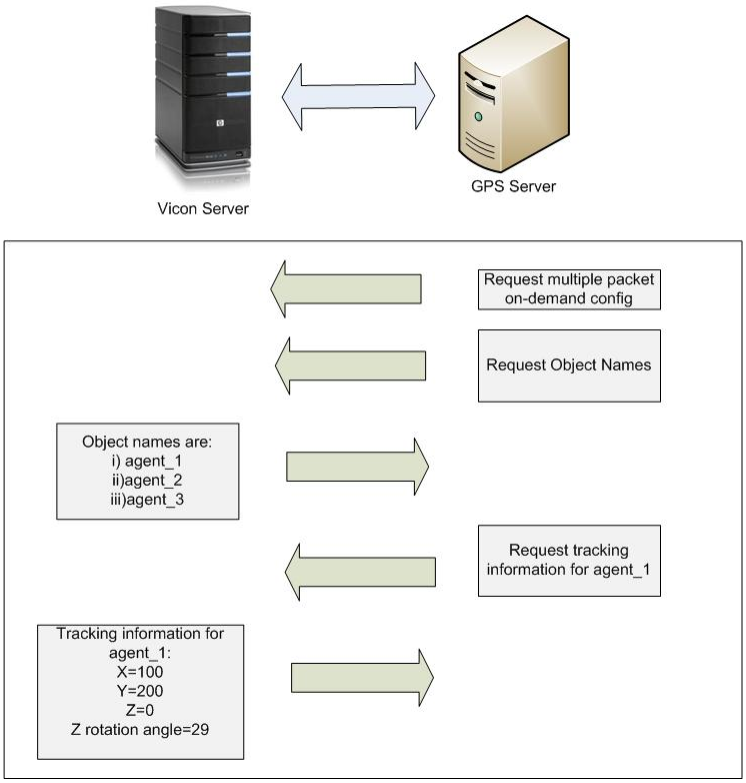


Figure 14: Process for requesting localization information from the Vicon server

A software interface was developed to allow the GPS server to request and acquire necessary tracking information from the Vicon server and to send that information to each of the agents. A TCP/IP connection will serve as a communications link between the GPS server and the Vicon server. The process for obtaining localization information is illustrated in Figure 14 and will be as follows. First, it is necessary to define how the Vicon server should respond to a query if the response is long enough to necessitate that multiple data packets be used. The Vicon server is configured to send additional packets relating to a response to a query upon request. This prevents the GPS server from being flooded with too many response packets at any given time.

The next step is to request the names of all the objects that are being tracked by the motion capture system. Once the names are received, for each of the objects that are detected, a request is sent to obtain available tracking information for that object. After the tracking information is received, it is parsed to extract relevant information required for the lab. This information, Cartesian coordinates of the object and Z-axis orientation, is stored in a data structure that will be later accessed to send localization information to the agents.

3.3.3 Agent Interface

3.3.3.1 Design considerations

The agents and the GPS server should use an agreed upon protocol that will allow the server to format the localization information in the data packet. Once the data packet is received by the agent, the agent will then know how to parse the packet to extract its position and orientation information. This is illustrated in Figure 15. Data will be sent to the agents via the wireless network protocol 802.11b. A design decision will need to be made regarding the client-server model that will be used.

One can allow the server to send data to the agents at regular intervals, allow the agents to request location information on demand, or enable both options. If a small subset of the agents require localization information, then an on-demand

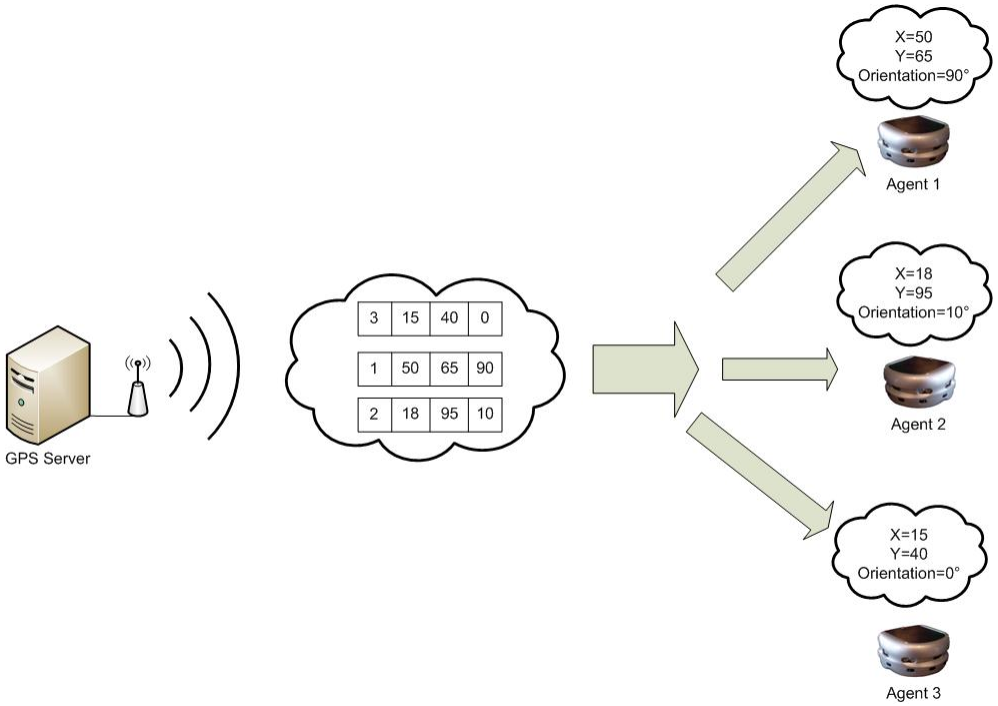


Figure 15: GPS server broadcasts localization information to all agents. Agents acquire only their localization information.

model is beneficial as network bandwidth is conserved by only sending out data to the agents who need it. An added benefit of using this model is that the agents who are apart of the subset will experience a significantly reduced latency period in between location updates. As the subset of agents needing location information grows, this model becomes increasingly inefficient as a consequence of the increased network overhead. This network overhead is attributed to the requests being sent to the server by each agent and the servers response to each of those requests. These requests and individual responses require a significant amount of data packets and bandwidth. A point is reached where it is more efficient to send information to all the agents on a periodic interval. With this model, there is no longer a need to burden the network with a large number of localization requests.

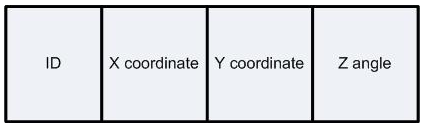


Figure 16: GPS format

3.3.3.2 Design decisions

The server will transmit localization information using a GPS protocol developed for this laboratory. This GPS protocol describes the order in which localization information will be packaged on the server side so that the agent can extract it upon receipt. Each GPS packet will contain the following fields in the listed order that are agent ID, x coordinate, y coordinate, and agent orientation in degrees (see Figure 16). Agents will be unable to request localization information on demand due to the significant network overhead that it can potentially require. Instead, the server will send out positioning and orientation information on periodic intervals (see Figure 15).

3.4 Summary of design

In this chapter, designs for each component of the laboratory infrastructure have been presented and discussed. The communications component was the first design to be presented. Designs for a communications protocol and a network architecture were described. A communications protocol provides the agents with a set of instructions to package and unpackaged data transmitted and received on the network. A network architecture defines the physical and logical network structure. The next design to be presented was navigation. This design provides the agents with a mechanism to generate a trajectory to reach a destination and a mechanism to guide and steer the agent along the generated trajectory. Last, a localization design was described. A method for acquiring localization information and distributing that information to agents was detailed in this design.

CHAPTER IV

TESTING

Since we are working with imperfect hardware in an environment that need not be predictable, it is often very difficult to evaluate how effective the proposed infrastructure design will be until we perform individual component tests. During these tests, we may discover that variables such as latency, noise, and hardware limitations affect our design adversely in ways that we did not predict or perhaps could not foresee. If test results indicate that our design does not meet the desired objectives, the design will be modified. This process of test and redesign can be iterative and will terminate only when the test results show that the design conforms to the specifications and objectives set forth. Each of the components in the laboratory infrastructure will undergo this test and redesign process.

4.1 Individual Component Testing

4.1.1 Communication

Before one can begin to evaluate the effectiveness of the communication protocol design, it is first necessary to test the actual communications hardware of the agent. This hardware primarily consists of the wireless network card, which cannot necessarily be assumed to function correctly. Each of the wireless cards of the agents were tested by running a script to send repeated pings to the agent. This script provided information that includes the number of packets dropped by the agent and the average time it took for the agent to send a response to the ping that was sent. The communications range of the network cards was also tested to verify that agents were able to communicate with each other from any location in the lab. In analyzing the test results, it was discovered that one of the agents had a faulty wireless card.

After verifying that the communications hardware was functioning correctly, the communications protocol was tested. A script was written that involved two agents sending messages to each other to observer whether or not they were able to package an outgoing message and to parse and extract information from an incoming data packet using the agreed upon protocol. An illustration of this test can be observed in Figure 17. The communications protocol test confirmed that the network infrastructure design was effective and that the overall communication component met specifications. Therefore, no design changes were necessary.

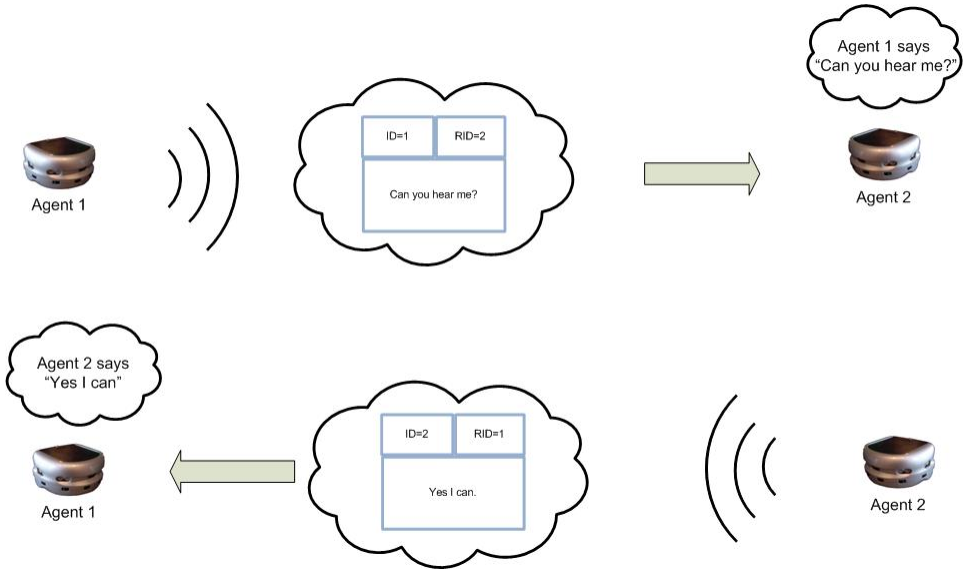


Figure 17: Communications test between two agents

4.1.2 Navigation

Unlike their personal computer counterparts, embedded systems, such as the khepera, do not have on-board floating point hardware to handle floating point calculations. Instead, they rely on floating point emulation software that can introduce significant round-off error, may not be able to handle very large floating point values, and can provide unreliable results. The localization data supplied to the path planner contains floating point values that will be used to calculate the destination orientation and the Euclidean distance of the destination. The path planner was first tested with integer

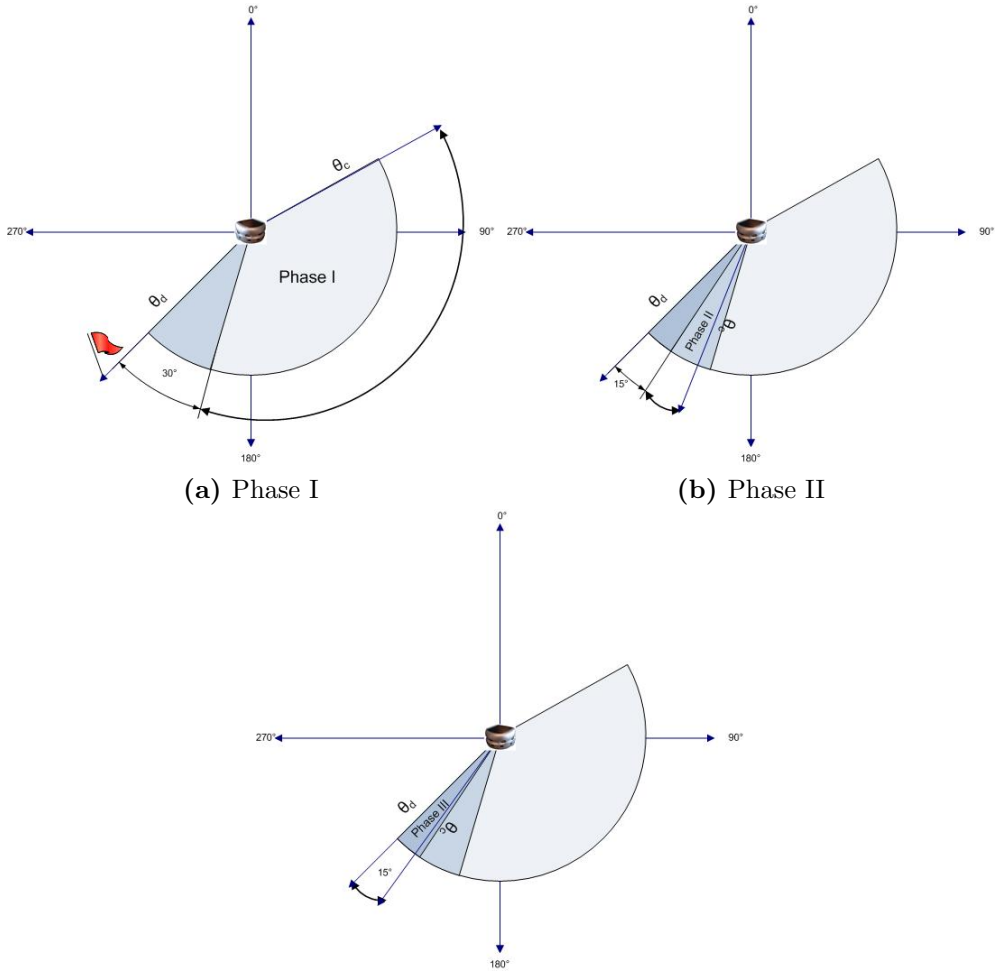
localization data to test whether the expected destination orientation and current distance matched the calculated results. Next, floating point data was inputted into the planner to analyze the error between results computed on the khepera versus a standard computer. There was no significant error observed during the comparisons, which verified that the emulation software was sufficient for the purposes of the lab.

The planner only requires the agent to perform turns and drive in a straight line. Therefore, tests were only conducted on the motion controllers ability to allow the agent to achieve a desired orientation. These tests consisted of choosing an arbitrary global orientation for the agent and passing that desired orientation to the motion planner. An unexpected result was that the desired orientation differed from the actual orientation consistently by more than 5 degrees. More investigation revealed that latency in the network was causing this issue to occur. Since localization information is sent to the agents over a wireless network, there can be significant delay between when the agents localization information is known and when the agent actually receives the information. This delay is precarious and can vary significantly.

In order to improve the orientation mechanism of the motion planner, the initial design presented in section 3.2.2 required modification to handle latency. A trivial but impractical solution would be to set the velocity of the robot while it is turning to achieve orientation to the lowest possible value. With this method, significant latency would have minimum and perhaps negligible impact on the agent's ability to reach a desired orientation. Although the described method solves the problem, it is impractical due to the time it would consume. An alternative method will be described below. It relies on the key observation that the agent's rotational speed is proportional to the accuracy in achieving a target orientation.

An important observation that guided the redesign is that accuracy becomes more of a priority as the agent gets closer to a target orientation. In the redesign, the orientation process is split into several phases (see Figure 18). In phase I, the agent

is assumed to be more than 30 degrees away from the desired orientation (θ_d) with respect to its current orientation (θ_c).The rotational speed of the agent is set to maximum in this phase. Once the agent is within 30 degrees of its desired target, phase II begins. In phase II, the velocity is decreased by half. The agent remains in this phase until it is within 15 degrees of the target. In the last phase, phase III, the velocity is again reduced by half. Empirical results show that the agents consistently obtain an error of less than 2 degrees when achieving orientation once the agent reaches phase III.



(a) Phase I **(b)** Phase II

(c) Phase III

Figure 18: An agent's rotation speed is set to maximum in phase I and is reduced by a factor of two in every ensuing phase.

4.1.3 Localization

Each of the subcomponents within the localization system was tested individually before the system was tested as a whole. The first subcomponent design to be tested was the motion capture configuration. In this test, the tracking system was evaluated to observe how well it was able to follow the agents as they moved throughout the lab. Although the Vicon system is capable of tracking agents with identical marker patterns, problems occurred if a potential "dead spot" exists in the lab. A dead spot is defined as an area where less than two cameras are covering it. In these dead spots, the system cannot track the agents. In many instances, the system will confuse the agent emerging from a dead spot with an agent close by. To solve this problem, the camera positions and orientations were changed so that the area covered by at least two cameras were maximized. Next, agents were restricted from being outside this area.

The next subcomponent that was tested was the motion capture interface. Localization data was requested by the GPS server and once responses were received by the GPS server, it was compared against the observed data from the Vicon server. This was done to ensure that the GPS server was properly requesting and receiving the correct information. The agent interface was the last subcomponent to be tested. Tests for this subcomponent involved sending test localization data to the agents to verify that they were able to receive data and that the agents were capturing the localization data that belonged to them. After the individual tests were conducted, the entire localization system was tested. Localization data was echoed to the server by the agent as the agent moved throughout the laboratory. This data was compared to the data reported by the localization system to verify accuracy.

4.2 Integrated Component Testing

In this section, an overview of the algorithm used to test the laboratory infrastructure will be presented. Next, it will be shown how the algorithm tests the infrastructure. Specific test procedures are then provided. Last, results of the test will be presented and also analyzed.

4.2.1 Gossip algorithm

A system wide test of the entire infrastructure of the multi-agent systems lab was conducted by executing an implemented version of the gossip algorithm [9] on the agents. This is a distributed algorithm where the agents send each other orientation information. Each agent then takes an average of all of the orientations that are received. Eventually, all of the agents will converge to the same orientation [9].

4.2.2 How the algorithm tests the infrastructure

The gossip algorithm is an excellent system wide test for this lab. It tests each of the laboratory components that are i) communication, ii) localization, and iii) navigation. Agents utilize the communication component to send and receive orientation information via the network. The orientation information is obtained from the localization component. Once an agent takes an average of the orientations that it receives, the next task is for the agent to rotate itself to achieve the average orientation. The navigation component is utilized to complete this task.

4.2.3 Test Procedures

Three khepera robots were used to test the laboratory infrastructure. A gossip algorithm was written in C using an integrated development environment (IDE). Before the algorithm was loaded, each khepera was manually rotated to a different orientation. The algorithm was then loaded on each of the kheperas and executed. During execution, kheperas periodically send out their orientation information. When a

khepera receives orientation information from two different kheperas, it calculates an average of its orientation and the two orientations that have been received. The khepera then rotates itself to the calculated average orientation. The algorithm repeats the above process until the kheperas converge to one orientation and then the algorithm terminates.

4.2.4 Results and analysis

The three kheperas were able to consistently converge to a mutual orientation. Convergence to an orientation was relatively fast. It took less than a minute on average. This fast convergence was expected and due to the small number of agents used. The above results verify that the laboratory infrastructure satisfied objectives and demonstrate that overall laboratory infrastructure design and implementation was a success.

CHAPTER V

LABORATORY USERS MANUAL

5.1 Equipment Setup and Configuration

Before the multi-agent systems laboratory is ready for use, there are three hardware components that will need to be setup and/or configured. They are the i) Vicon Motion Capture System, ii) Khepera robots, and iii) GPS server. Due to the complex nature of the motion capture system, it is extremely important that this system is carefully setup and configured. If care is not taken, inaccurate localization information will propagate through the system and may adversely affect the performance of the laboratory.

5.2 Vicon Motion Capture System

5.2.1 Initializing the System

Step 1: Turn on the DataStation

Step 2: Wait for the boot-up sequence to finish.

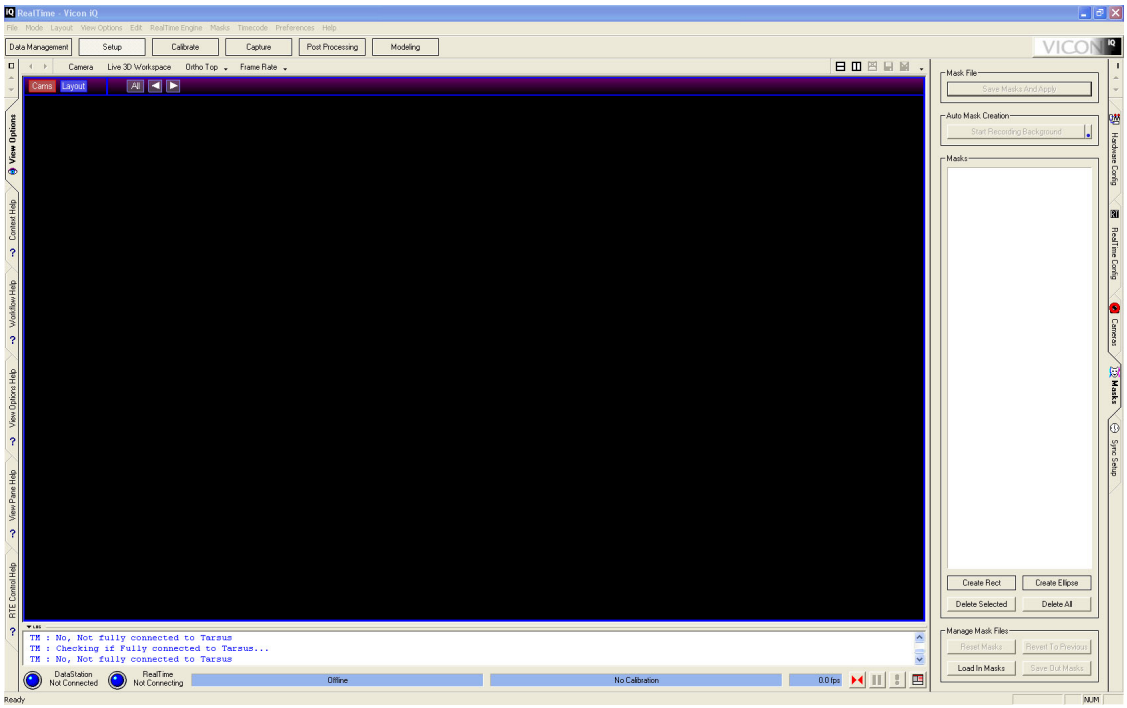


Figure 19: Vicon iQ software interface

Step 3: Launch the Vicon iQ software interface (see Figure 19).

Step 4: Connect to the DataStation.

5.2.2 Calibration

To ensure that the tracking data reported by the system is accurate, the following calibration process should be followed:

Step 1: Click on the menu item titled "Calibration" on the software interface (see Figure 19). This will display all the necessary software tools that will be needed for the calibration process.

Step 2: Visually inspect that the entire roaming space is seen by at least two cameras. Using the wand provided in the calibration kit (see Figure 5c) that was included with the system, the user can move the wand to a desired area. The user can then check to see that the markers on the wand are shown on the camera output display of the software as demonstrated in Figure 20.

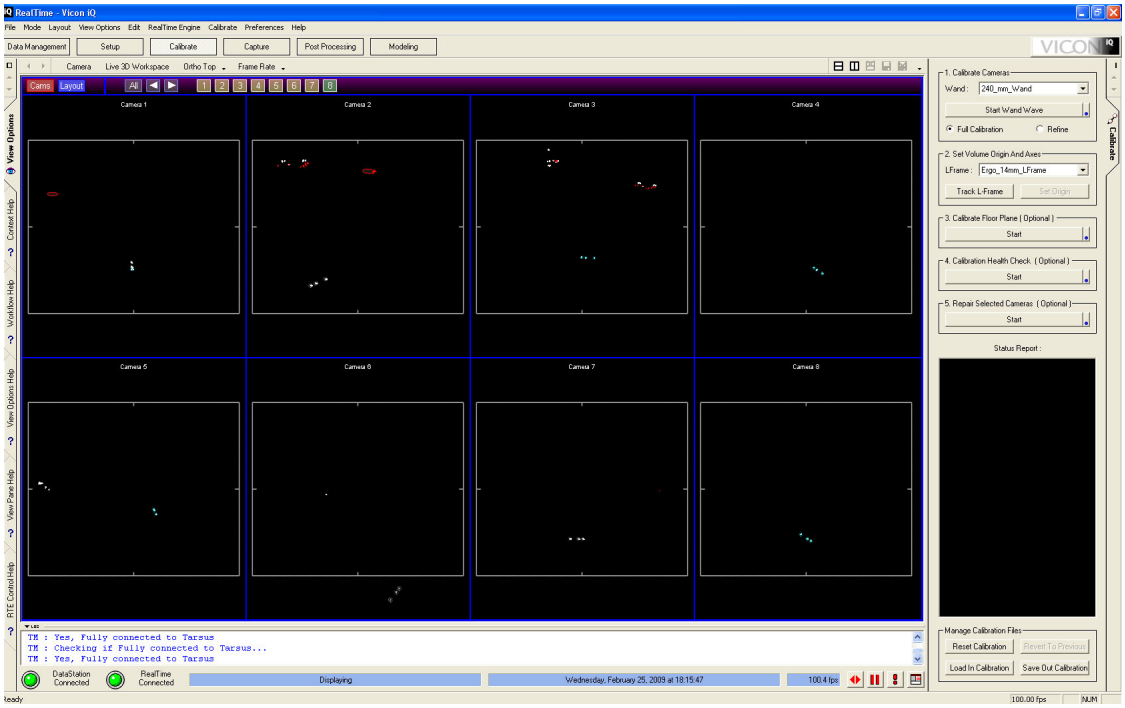


Figure 20: Observe that at the wand's current position, it can be seen by every camera except for camera 6.

Step 3: Click the "Start Wand Wave" Icon, which should be available under the Calibration menu. The user should begin waving the wand around the desired agent tracking area. The status report window will provide information such as the number of data points that have been currently collected. After a sufficient number of data points have been collected and the camera output windows indicate that the desired tracking area has been covered, the process can be stopped by clicking the "stop wave " icon.

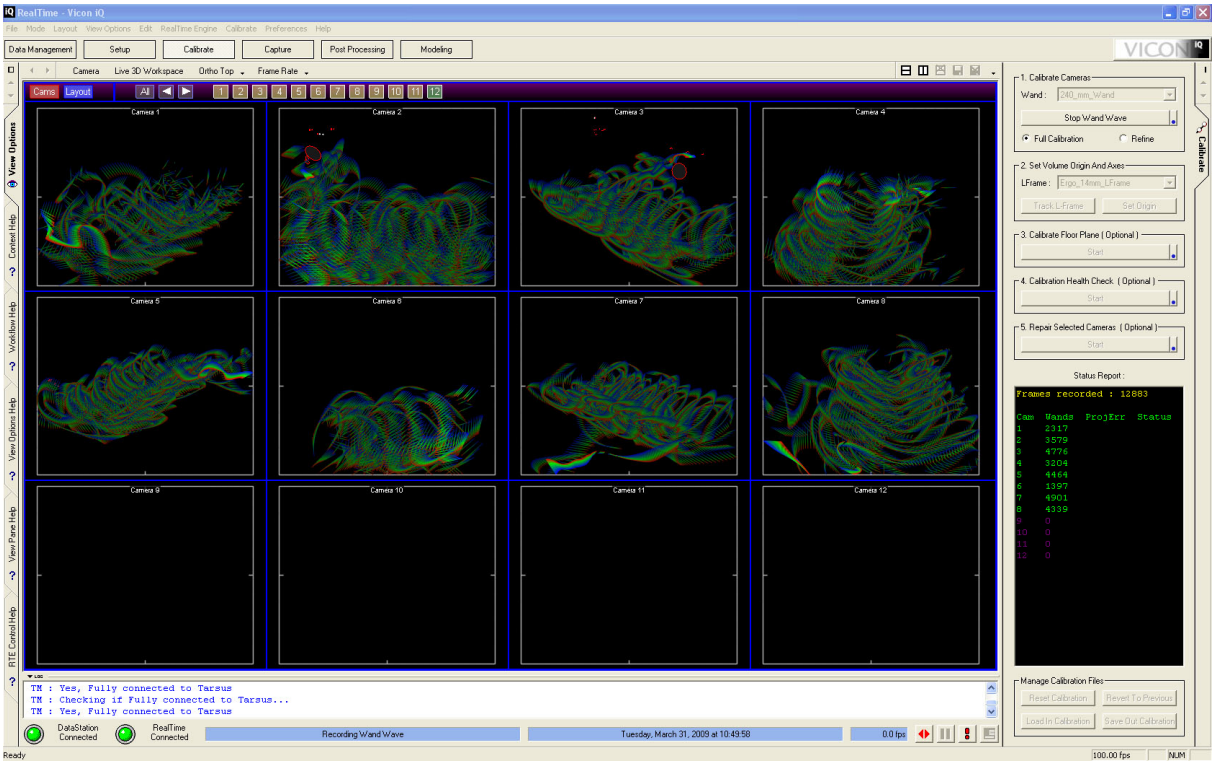
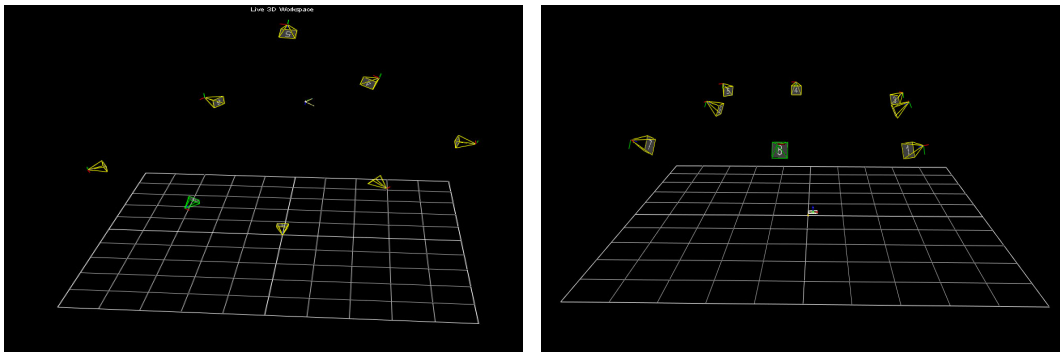


Figure 21: Snapshot of the system being calibrated

Step 4: Once the cameras have been calibrated, the software will tell the user how well each of the cameras are calibrated as shown in Figure 22. If a camera's result is fair or worse, repeat the calibration process and capture more data points.

Figure 22: Each of the cameras have a calibration status of good or excellent. This confirms that the calibration process is a success.

Step 5: Use the base plate provided with the system to set the global reference point for all of the cameras. Figure 23a shows a 3-d representation of what the cameras are collectively capturing before the origin is set. The base plate appears to be floating in space, when it is actually setting on the floor of the lab. The user can define this base plate as the origin and the ground plane. Figure 23b demonstrates the results of setting the origin. The base plate now defines the origin and ground plane.

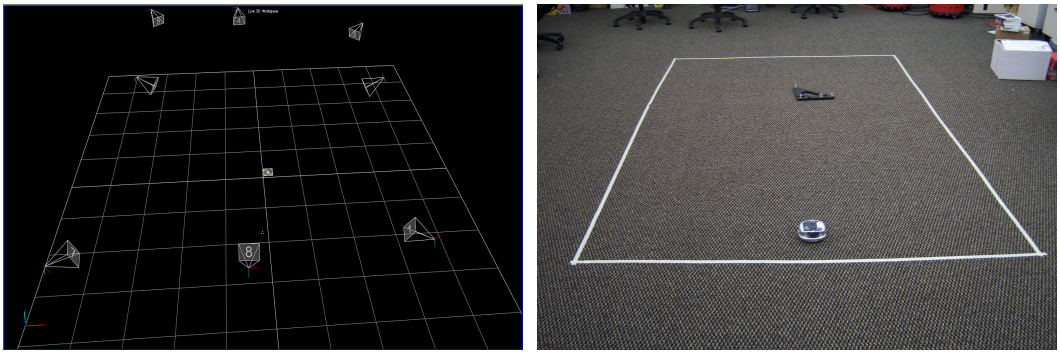


(a) before setting reference **(b)** after setting reference

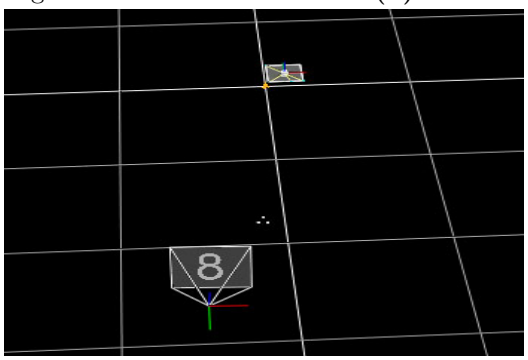
Figure 23: Setting the global reference point

5.2.3 Creating agent tracking objects

The process for creating an agent tracking object is below. Note that each agent has 3 reflective markers on its top surface that form a triangle. This can be observed in Figure 13. Figure 24c provides a close up view of the triangular markers as captured by the motion capture system, while Figure 24b is an actual photograph of the robot and the markers in the laboratory.



(a) before setting reference **(b)** laboratory photograph



(c) Close-up of agent in 24a

Figure 24: Setting the global reference point

Step 1: Click on the menu tab labeled "**Capture**" on the iQ software interface (see Figure 19).

Step 2: Find the side bar menu. Click on the tab labeled "**Create/Edit Objects**".

Step 3: Press the button labeled "**Begin editing**".

Step 4: Use the mouse to drag a rectangle over the three markers as demonstrated in Figure 25.

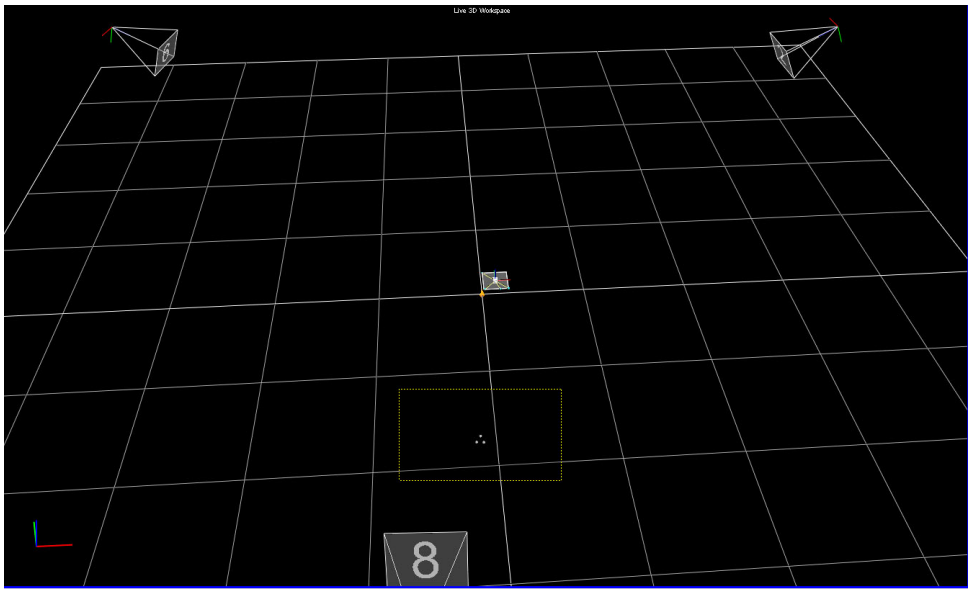


Figure 25: A yellow rectangular box is created by dragging the mouse over the desired markers.

Step 5: Press the "**Create Object**" button in side menu. The 3 markers will now be recognized as one object as shown in Figure 26.

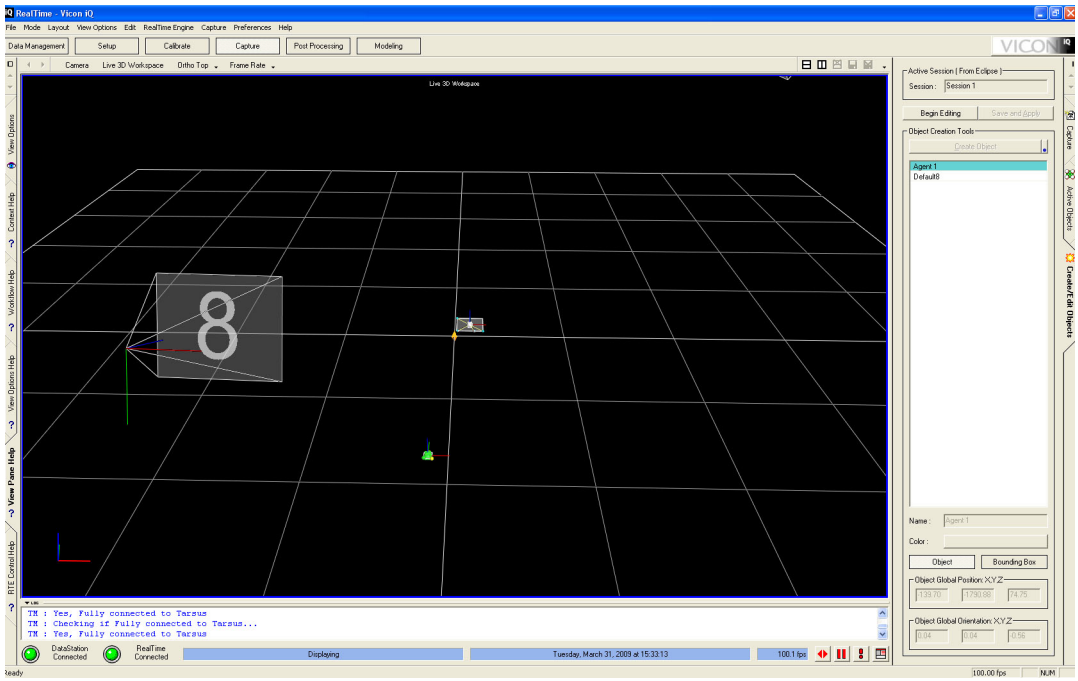


Figure 26: Agent object created

5.3 Server

For testing and code modification purposes, the software code will be executed in Visual Studio 2008 in debug mode on the server. The server will be assumed to have already been connected to the wireless ad-hoc network that the agents are on.

Step 1: Launch the visual studio solution "**Khepera_GPS.sln**".

Step 2: Select the "**Debug**" menu item and press the "**start debug**" option (see Figure 27) or alternatively press the "**F5**" key to launch the software in debug mode.

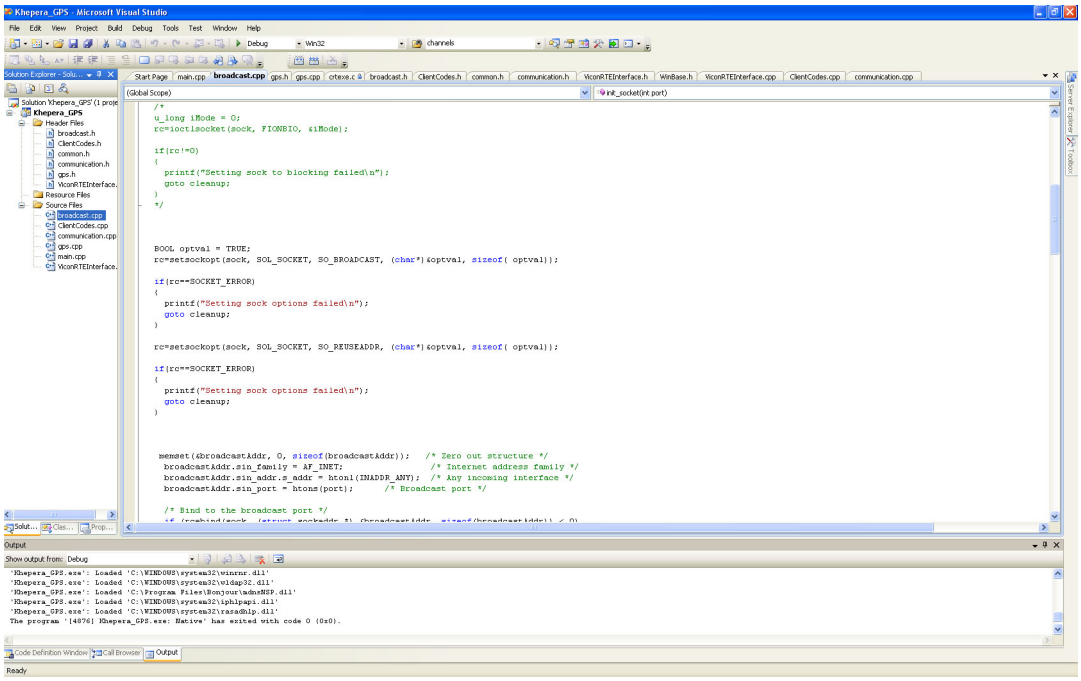
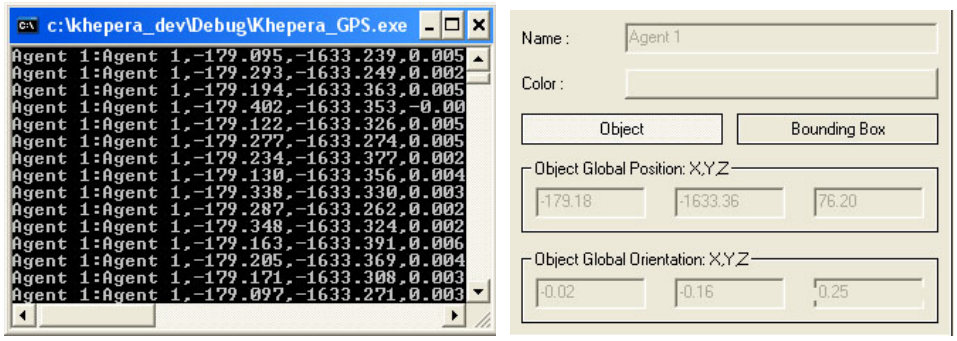


Figure 27: Khepera_GPS solution in Visual Studio 2008

Step 3: Verify that the localization information for the agent displayed by the Khepera_GPS program (Figure 28a) is identical to the information presented on the iQ software interface (Figure 28b).



(a) GPS Server Localization display **(b)** Vicon iQ software display

Figure 28: The localization data displayed in Figures 28a) and 28b) are approximately the same, which verifies that the GPS server is operating correctly.

5.4 Khepera

Since the kheperas are running a Linux kernel, configuring their wireless settings for ad-hoc mode and their ssid to some predefined id is a straightforward process. It is assumed that these steps have been already completed. Programming the kheperas to run a distributed algorithm is an important task that will be repeatedly performed, therefore it is critical that the user understand how to use the integrated development environment (IDE). The IDE is a tool thats provides the developer with a means to develop, compile, and build software for the khepera platform. After a script is generated using the IDE, it is necessary to connect to the khepera so that the script can be transfered an executed.

5.4.1 Developing Software

Step 1: Launch Code Blocks, which is the Khepera IDE.(see Figure 29).

Step 2: Create a console project and select C source as the file option.

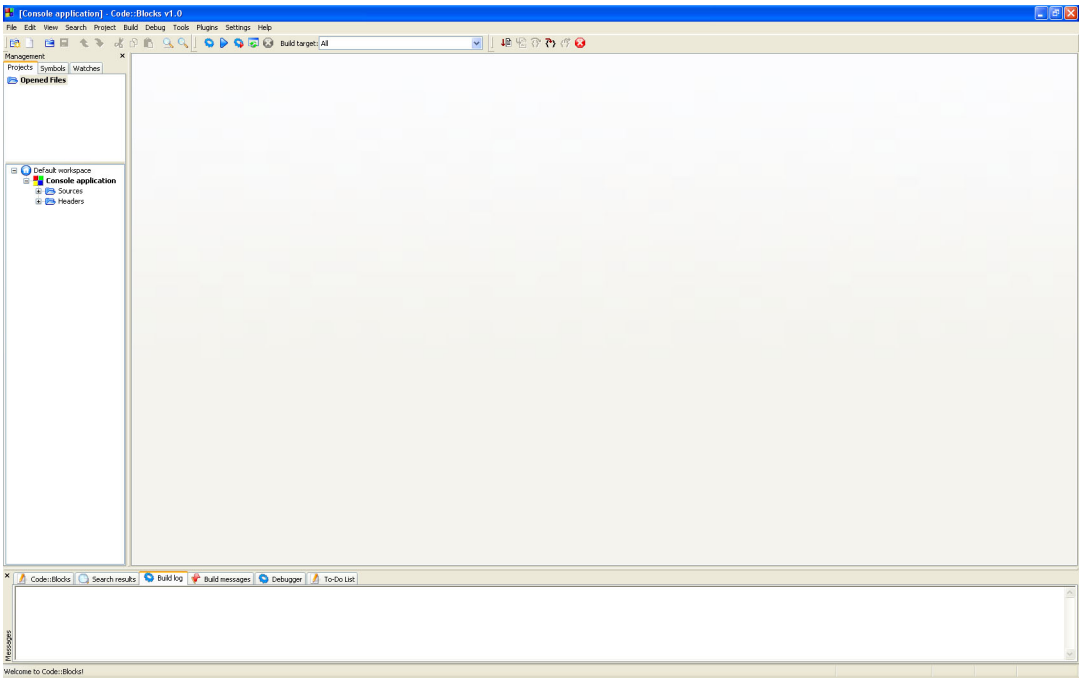


Figure 29: Code Blocks (Khepera IDE)

Step 3: **IMPORTANT!!** Change the Selected compiler to **cross_korebot** in the build options menu (see Figure 30).

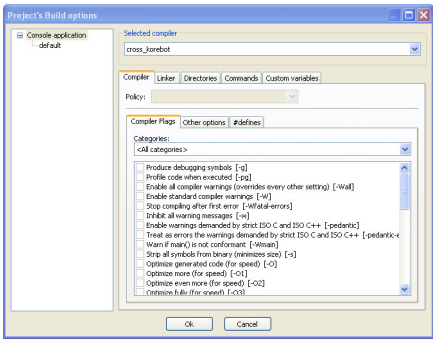


Figure 30: Selected compiler option displays **"cross_korebot"**

Step 4: Create and add the desired C source files to the project.

Step 5: Compile and build code by pressing the keys **Ctrl-F9**.

5.4.2 Transferring and Executing script

Step 1: Launch a secure socket host (SSH) shell.

Step 2: Connect to the khepera robot by entering in it's IP address under the host field and the user name. The default user name for each for all of the kheperas is "root". An example of this can be observed in Figure 31.

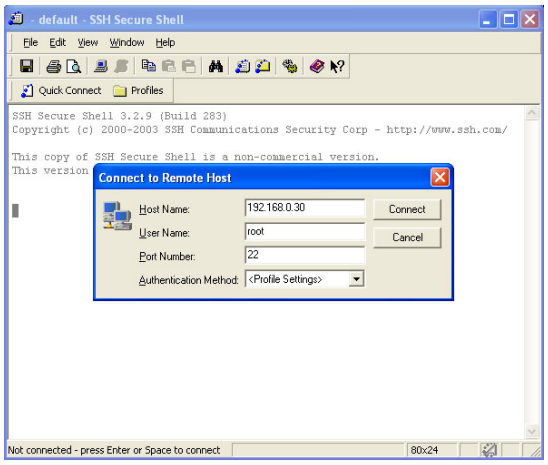


Figure 31: Observe that the host name(IP address) is "**192.168.0.30**" and the user name is "**root**".

Step 3: Locate the executable file on development computer that one wishes to transfer by browsing the directory. Once the file is located, drag it from the current directory to the main directory of the khepera on the right.

Step 4: Execute the file by typing in ".\" followed by the name of the executable in the shell command prompt.

CHAPTER VI

CONCLUSION

6.1 Summary

In this thesis, results of a laboratory hardware investigation were presented that included an analysis of each hardware component under consideration. A design and a corresponding design analysis were then provided for each of the laboratory infrastructure components that are i) communication, ii) navigation, and iii) localization. During testing, it was discovered that modifications to the design of the navigation component were necessary to improve overall laboratory performance. A new design was proposed and tested for this component. After testing the individual components, and integration test was completed. The integration test consisted of implementing a gossip algorithm to evaluate how the components of the infrastructure performed as a whole. Results of the test were provided. These results demonstrate that the design satisfied each of the overall laboratory objectives.

6.2 Future Recommendations

Laboratory infrastructure designs presented in this thesis were intended to provide a foundation that would allow the multi-agent systems laboratory to improve and expand. Much of the laboratory development was focused on developing a basic design to demonstrate that the current infrastructure and hardware were sufficient to build a laboratory. Since this paper has established the sufficiency of the infrastructure and hardware for a small set of agents, the next logical progression is to modify the design to handle a large number of agents (30+). Below, a list of suggested design

modifications are provided, which will help facilitate the goal of enabling the multi-agent systems laboratory to scale well.

1. *Implement an obstacle avoidance mechanism for the agents.* If one is to introduce a large number of agents into the laboratory, an obstacle avoidance mechanism is necessary. If there are n agents, there are at least $n - 1$ potential static and mobile objects that an agent will need to consider while navigating to a desired destination.

2. *Modify the localization distribution mechanism.* Currently, each agent's localization information is sent out individually and is wrapped in a data packet. A more efficient method would consist of wrapping more than one agent's localization in the data packet. This would reduce latency in receiving the information and would also reduce the network utilization, while conserving bandwidth.

3. *Automate agent mapping process.* As mentioned in the thesis, it is necessary to manually map/associate each agent with its corresponding object representation that the Vicon system tracks. With a large number of agents, this manual process can become very tedious, cumbersome, and time consuming. Furthermore, this mapping must be done each time the motion capture system is started. An automated mapping process will greatly reduce the time it takes to initialize the laboratory and the frustration that results in manually associating each agent with an object.

REFERENCES

[1] Yuefeng Liu and A. Zhang. "Multi-agent System and Its Application in Combat Simulation," *IEEE Computational Intelligence and Design*, October 2008.

[2] Albert Ko, Hery Lau, and Rex Sham."Application of Distributed Wireless Sensor Network on Humanitarian Search and Rescue Systems," *IEEE FGCN*, December 2008.

[3] Uichin Lee, Eugenio Magistretti, and Mario Gerla. "Dissemination and harvesting of Urban Data Using Vehicular Sensing Platforms," *IEEE Transactions on Vehicular Technology*, February 2009.

[4] "The khepera III robot", December 5, 2006. [Online]. Available: http://www.k-team.com. [Accessed: March 20, 2008].

[5] Balakrishnan, H., et al. "Lessons from Developing and Deploying the Cricket Indoor Location System," MIT CSAIL, November 7, 2003.

[6] Anthony Row, Charles Rosenberg, and Illah Nourbakhsh. "A Second Generation Low Cost Embedded Color Vision System," *IEEE Computer Vision and Pattern Recognition*, April 2005.

[7] "Vicon motion capture system," January 12, 2009. [Online]. Available: http://www.vicon.com. [Accessed: February 25, 2009].

[8] Yanjie Liu, Lining Sun, and Zhenweian An. "An Approach for Generationg High Velocity and High Acceleration Trajectories of Industrial Robots," *IEEE Computational Intelligence in Robotics and Automation*, June 2005.

[9] Tuncer C. Aysal, Mehmet E. Yildiz, and Anna Scaglione. "Broadcast Gossip Algorithms," *IEEE Information Theory Workshop*, May 2008.

APPENDIX A

SOURCE CODE LIBRARY

A.1 GPS Server

initializeViconRTE() Initializes a TCP/IP connection with the Vicon real-time engine server so that the GPS server can request and retrieve localization information

getRTEData() Sends requests to the Vicon server for localization information of all objects. Once the Vicon server responds to the requests, the information received by the GPS server is parsed and organized into data structures that can be later accessed.

broadcastGPS() Accesses the data structure that stores the agents localization information. Each agents GPS information is broadcast over the wireless network using UDP.

A.2 Khepera

A.2.1 Communication

sendgenericMessage() Sends a message to another agent.

recvgenericMessage() Enables agent to receive a message that has been broadcast by another agent.

msgparser() Parses message received by an agent to extract relevant information.

A.2.2 Navigation

rotatetoDegree() Rotates the agent to a desired global orientation.

gotoCoordinate() Given an appropriate Cartesian coordinate, the function allows agent to reach that coordinate.

A.2.3 Localization

getGPSposition() Acquires GPS position by listening to the GPS Server broadcast. Information broadcasted by GPS server is parsed to extract localization data.

initializeGPS() Initializes the UDP socket to allow for GPS broadcasts to be received by the agent.